Racket Programming Assignment #2:
Racket Functions & Recursion

Thomas Moskal

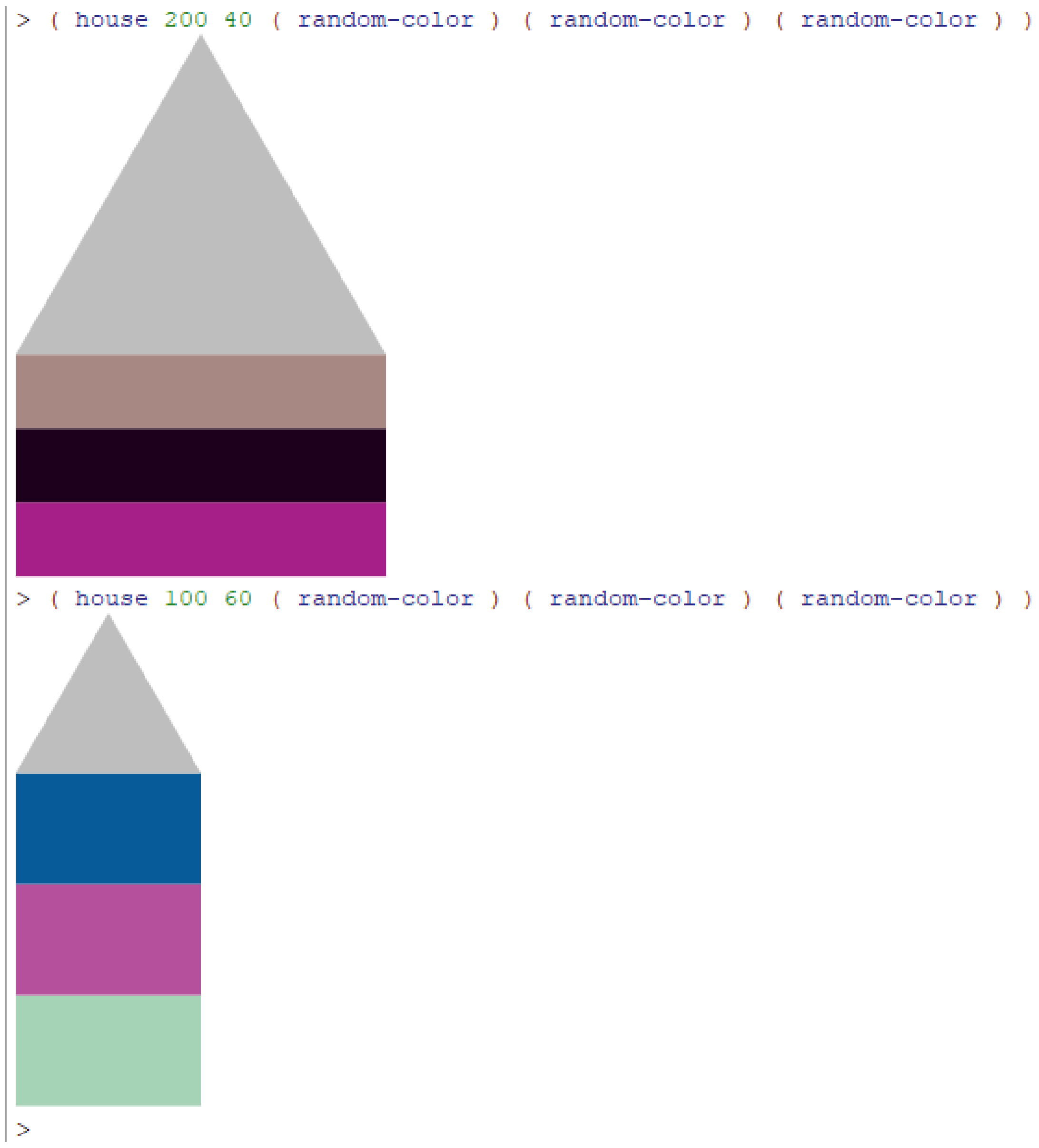
CSC 344 – Professor Graci

Due date: 2/25/22

Learning Abstract:

This assignment is a continuation of our practice in Racket. The focus of this assignment is on functions but, most of the examples are about Recursion based functions. This assignment also shows the flexibility of the 2htdp/image library. It's interesting to see how a few lines of code can drastically alter an image and make it into something completely different. I especially liked the creation portion at the end of this assignment. I made a rhombus fractal that has many layered colors.

Task 1 – Colorful Permutations of Tract Houses

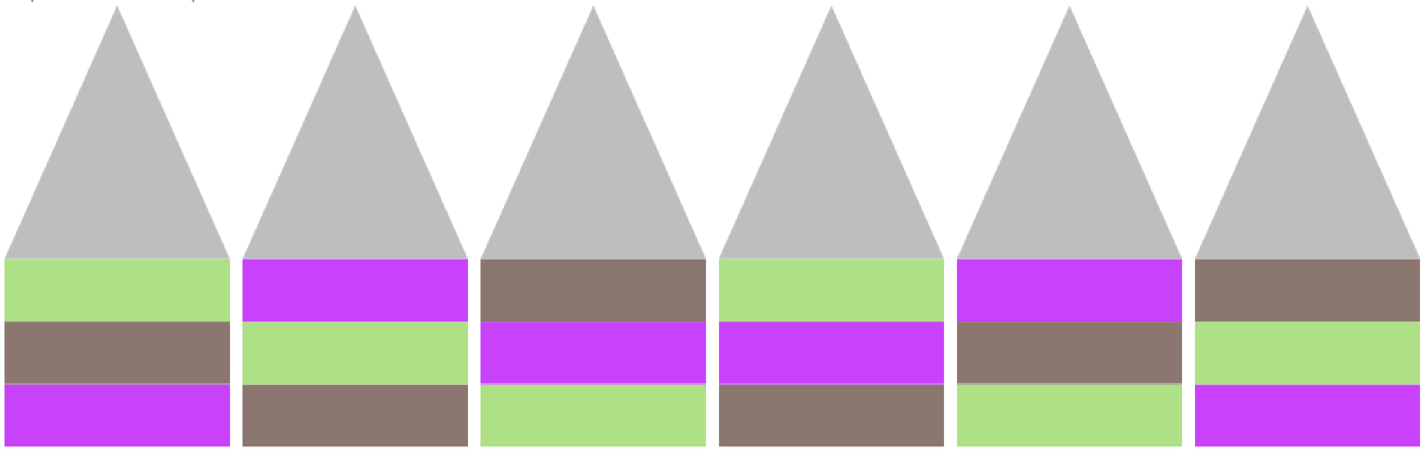


```
> ( house 200 40 ( random-color ) ( random-color ) ( random-color ) )
```

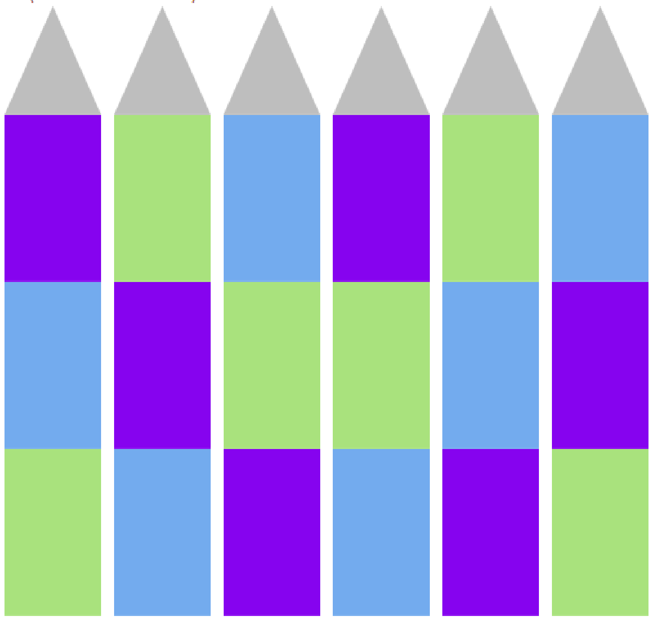
```
> ( house 100 60 ( random-color ) ( random-color ) ( random-color ) )
```

```
>
```

```
> ( tract 700 150 )
```



```
> ( tract 300 400 )
```



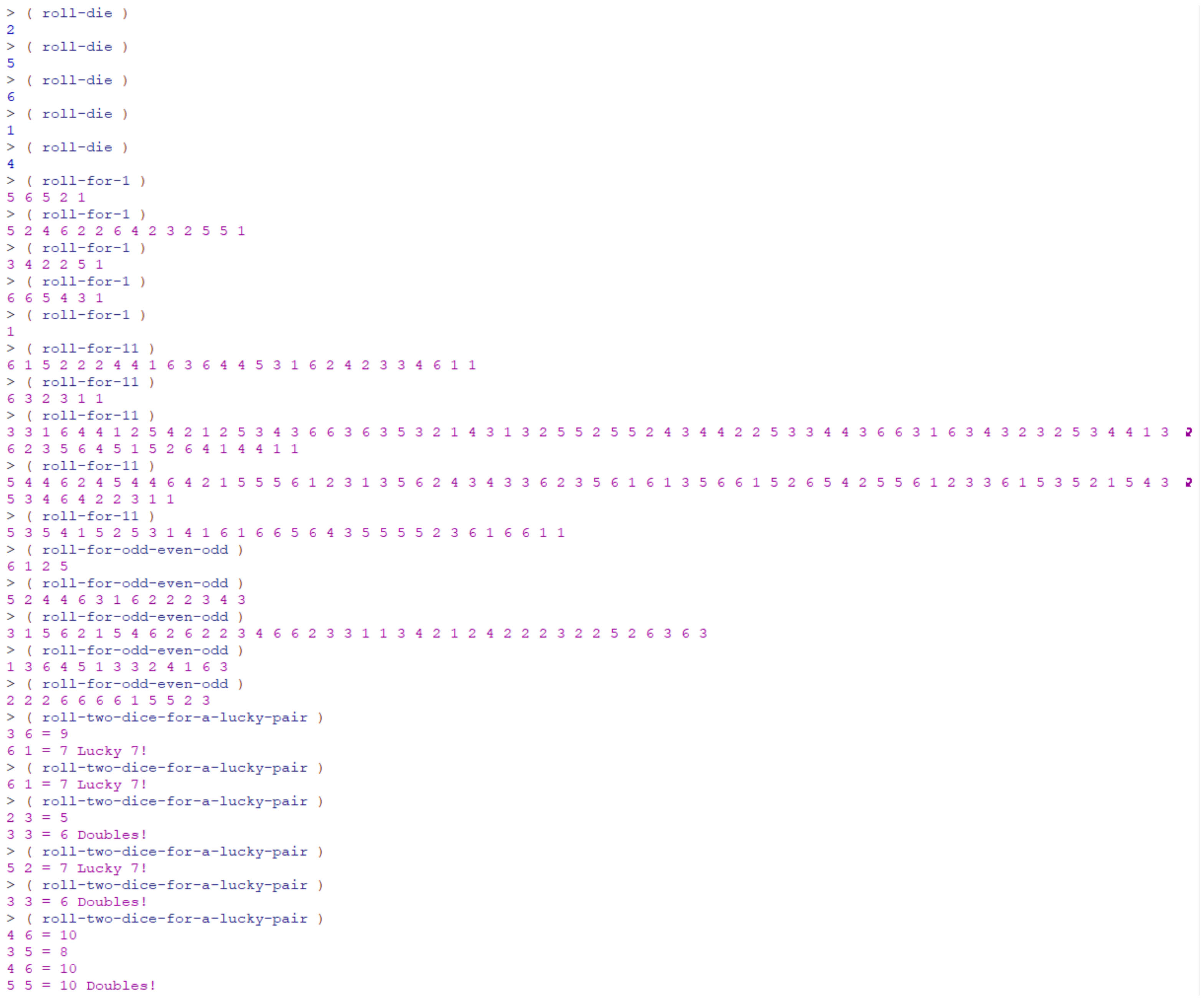
```
>
```

In this task #1 part one I emulated the examples shown by the professor. I created a house, and then a function to create multiples of that house. The assignment only called for a specific number of houses so it is not as flexible as it could be.

```
#lang racket

( require 2htdp/image )

( define ( rgb-value ) ( random 256 ) )
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )

( define ( house widths heights third second first)
   ( define  width   ( / widths 4 ) )
   ( define  height   ( / heights 4 ) )
   ( define roof (triangle widths "solid" "gray"))
   ( define floor1 (rectangle widths heights "solid" first))
   ( define floor2 (rectangle widths heights "solid" second))
   ( define floor3 (rectangle widths heights "solid" third))
   ( define house1 (above roof floor1 floor2 floor3))
   house1
)

( define ( tract widths heights)
   ( define  width   ( / widths 4 ) )
   ( define  spacer-width  10 )
   ( define  height   ( / heights 4 ) )
   ( define roof (triangle width "solid" "gray"))
   ( define floor1 (rectangle width height "solid" (random-color)))
   ( define floor2 (rectangle width height "solid" (random-color)))
   ( define floor3 (rectangle width height "solid" (random-color)))
   ( define space (rectangle spacer-width height "solid" "white"))
   ( define house1 (beside (above roof floor1 floor2 floor3) space))
   ( define house2 (beside (above roof floor3 floor1 floor2) space))
   ( define house3 (beside (above roof floor2 floor3 floor1) space))
   ( define house4 (beside (above roof floor1 floor3 floor2) space))
   ( define house5 (beside (above roof floor3 floor2 floor1) space))
   ( define house6 (beside (above roof floor2 floor1 floor3) space))
   ( define da_block (beside house1 house2 house3 house4 house5 house6 ))
   da_block
)
```

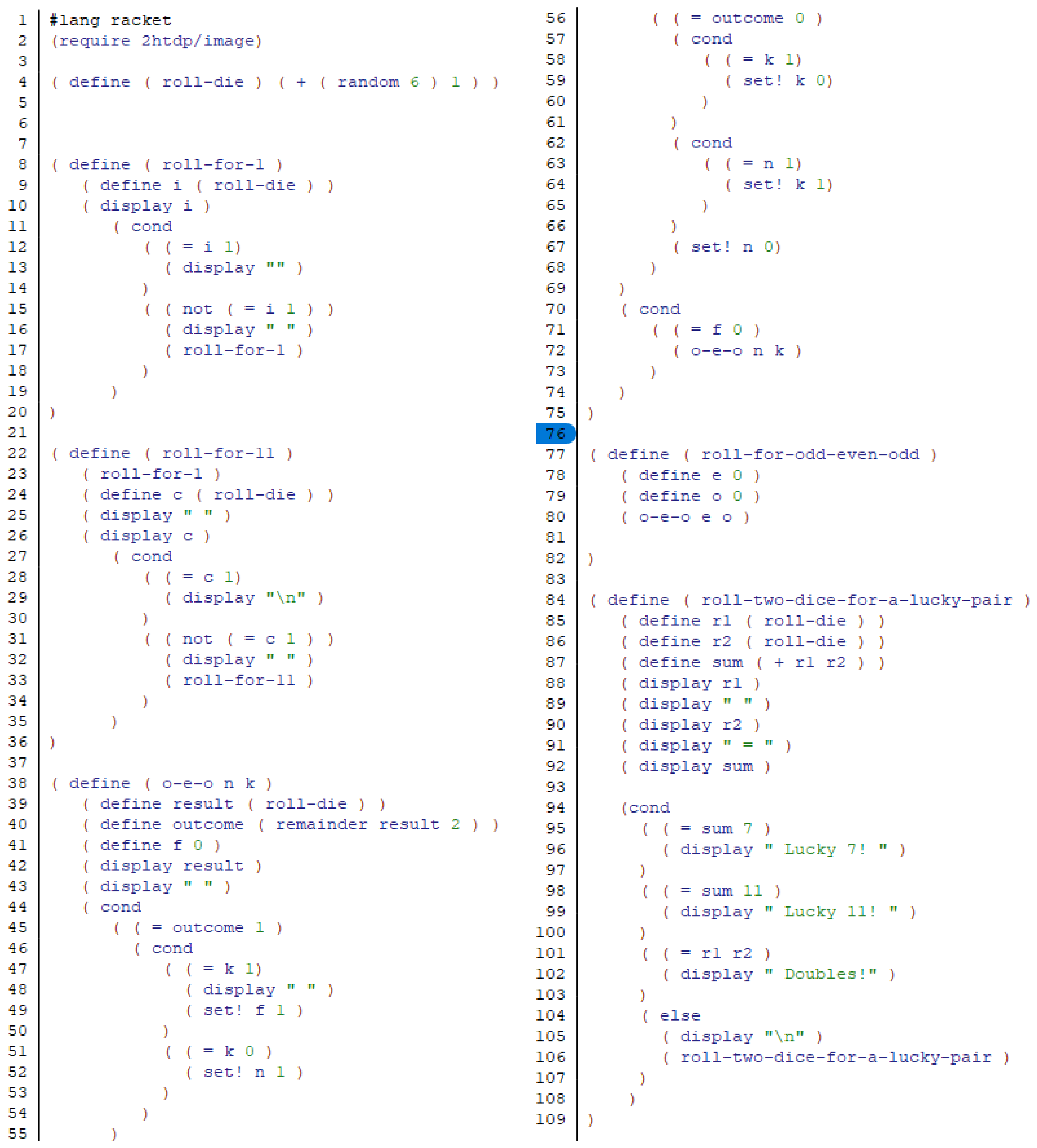
As you can see, the code is linear but useful. If one made changes to the permutation portion of the tract function, then it could be more randomized in its patterning.

Task 2 – Dice



```
> ( roll-die )
2
> ( roll-die )
5
> ( roll-die )
6
> ( roll-die )
1
> ( roll-die )
4
> ( roll-for-1 )
5 6 5 2 1
> ( roll-for-1 )
5 2 4 6 2 2 6 4 2 3 2 5 5 1
> ( roll-for-1 )
3 4 2 2 5 1
> ( roll-for-1 )
6 6 5 4 3 1
> ( roll-for-1 )
1
> ( roll-for-11 )
6 1 5 2 2 2 4 4 1 6 3 6 4 4 5 3 1 6 2 4 2 3 3 4 6 1 1
> ( roll-for-11 )
6 3 2 3 1 1
> ( roll-for-11 )
3 3 1 6 4 4 1 2 5 4 2 1 2 5 3 4 3 6 6 3 6 3 5 3 2 1 4 3 1 3 2 5 5 2 5 5 2 4 3 4 4 2 2 5 3 3 4 4 3 6 6 3 1 6 3 4 3 2 3 2 5 3 4 4 1 3
6 2 3 5 6 4 5 1 5 2 6 4 1 4 4 1 1
> ( roll-for-11 )
5 4 4 6 2 4 5 4 4 6 4 2 1 5 5 5 6 1 2 3 1 3 5 6 2 4 3 4 3 3 6 2 3 5 6 1 6 1 3 5 6 6 1 5 2 6 5 4 2 5 5 6 1 2 3 3 6 1 5 3 5 2 1 5 4 3
5 3 4 6 4 2 2 3 1 1
> ( roll-for-11 )
5 3 5 4 1 5 2 5 3 1 4 1 6 1 6 6 5 6 4 3 5 5 5 5 2 3 6 1 6 6 1 1
> ( roll-for-odd-even-odd )
6 1 2 5
> ( roll-for-odd-even-odd )
5 2 4 4 6 3 1 6 2 2 2 3 4 3
> ( roll-for-odd-even-odd )
3 1 5 6 2 1 5 4 6 2 6 2 2 3 4 6 6 2 3 3 1 1 3 4 2 1 2 4 2 2 2 3 2 2 5 2 6 3 6 3
> ( roll-for-odd-even-odd )
1 3 6 4 5 1 3 3 2 4 1 6 3
> ( roll-for-odd-even-odd )
2 2 2 6 6 6 6 1 5 5 2 3
> ( roll-two-dice-for-a-lucky-pair )
3 6 = 9
6 1 = 7 Lucky 7!
> ( roll-two-dice-for-a-lucky-pair )
6 1 = 7 Lucky 7!
> ( roll-two-dice-for-a-lucky-pair )
2 3 = 5
3 3 = 6 Doubles!
> ( roll-two-dice-for-a-lucky-pair )
5 2 = 7 Lucky 7!
> ( roll-two-dice-for-a-lucky-pair )
3 3 = 6 Doubles!
> ( roll-two-dice-for-a-lucky-pair )
4 6 = 10
3 5 = 8
4 6 = 10
5 5 = 10 Doubles!
```

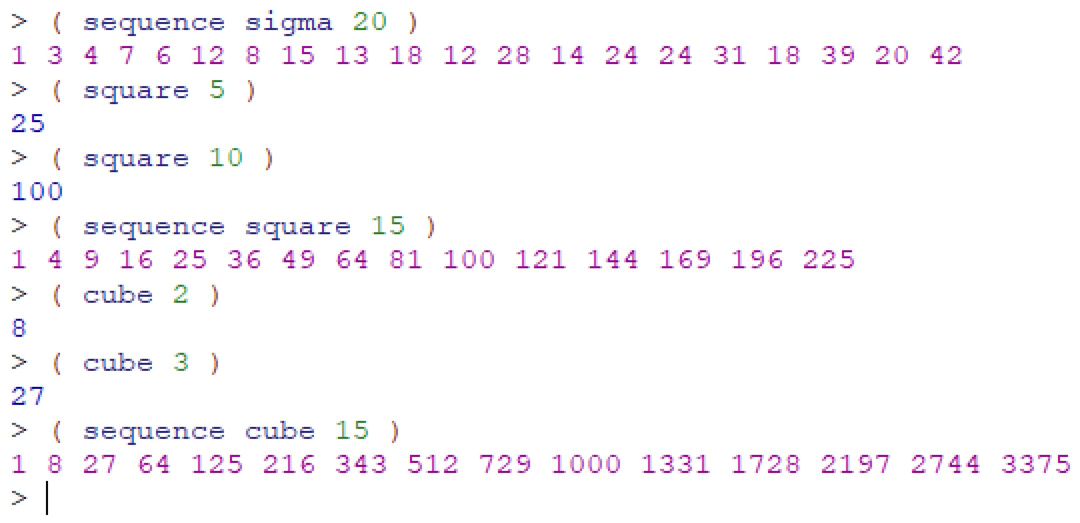
In this task #2 we created code that emulates a random dice roll. Then we used that function to study recursive functions. The dice roll is good for recursion examples due to it's randomness.



```racket
#lang racket
(require 2htdp/image)

( define ( roll-die ) ( + ( random 6 ) 1 ) )



( define ( roll-for-1 )
   ( define i ( roll-die ) )
   ( display i )
      ( cond
         ( ( = i 1)
           ( display "" )
         )
         ( ( not ( = i 1 ) )
           ( display " " )
           ( roll-for-1 )
         )
      )
)

( define ( roll-for-11 )
   ( roll-for-1 )
   ( define c ( roll-die ) )
   ( display " " )
   ( display c )
      ( cond
         ( ( = c 1)
           ( display "\n" )
         )
         ( ( not ( = c 1 ) )
           ( display " " )
           ( roll-for-11 )
         )
      )
)

( define ( o-e-o n k )
   ( define result ( roll-die ) )
   ( define outcome ( remainder result 2 ) )
   ( define f 0 )
   ( display result )
   ( display " " )
   ( cond
      ( ( = outcome 1 )
        ( cond
           ( ( = k 1)
             ( display " " )
             ( set! f 1 )
           )
           ( ( = k 0 )
             ( set! n 1 )
           )
         )
      )
      ( ( = outcome 0 )
        ( cond
           ( ( = k 1)
             ( set! k 0)
           )
        )
        ( cond
           ( ( = n 1)
             ( set! k 1)
           )
        )
        ( set! n 0)
      )
   )
   ( cond
      ( ( = f 0 )
        ( o-e-o n k )
      )
   )
)

( define ( roll-for-odd-even-odd )
   ( define e 0 )
   ( define o 0 )
   ( o-e-o e o )

)

( define ( roll-two-dice-for-a-lucky-pair )
   ( define r1 ( roll-die ) )
   ( define r2 ( roll-die ) )
   ( define sum ( + r1 r2 ) )
   ( display r1 )
   ( display " " )
   ( display r2 )
   ( display " = " )
   ( display sum )

   (cond
     ( ( = sum 7 )
       ( display " Lucky 7! " )
     )
     ( ( = sum 11 )
       ( display " Lucky 11! " )
     )
     ( ( = r1 r2 )
       ( display " Doubles!" )
     )
     ( else
       ( display "\n" )
       ( roll-two-dice-for-a-lucky-pair )
     )
    )
)
```

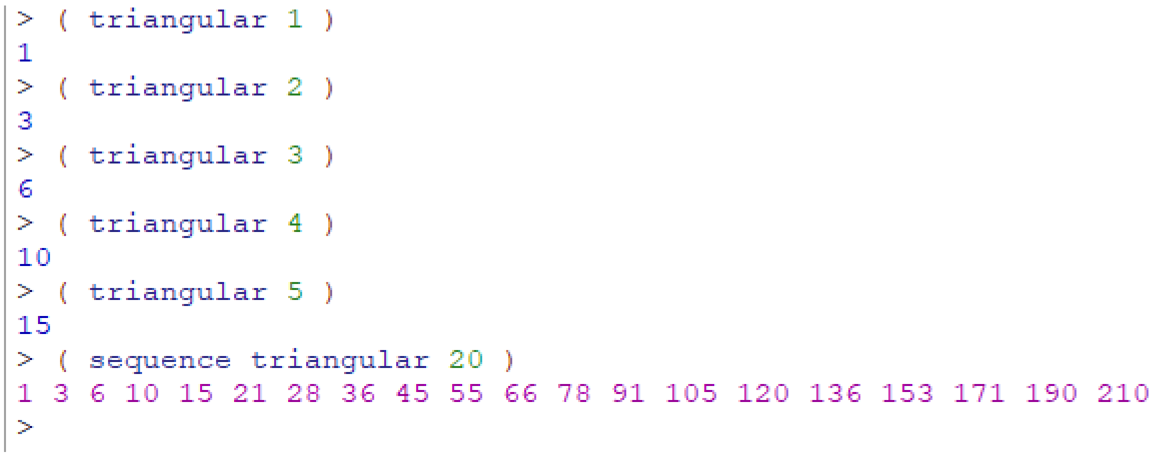
<u>While not exactly the perfect set up for the odd-even-odd function I was able to fulfill the requirements of the task. I think there may have been a better way to use recursion in that function.</u>

Task 3 – Number Sequences



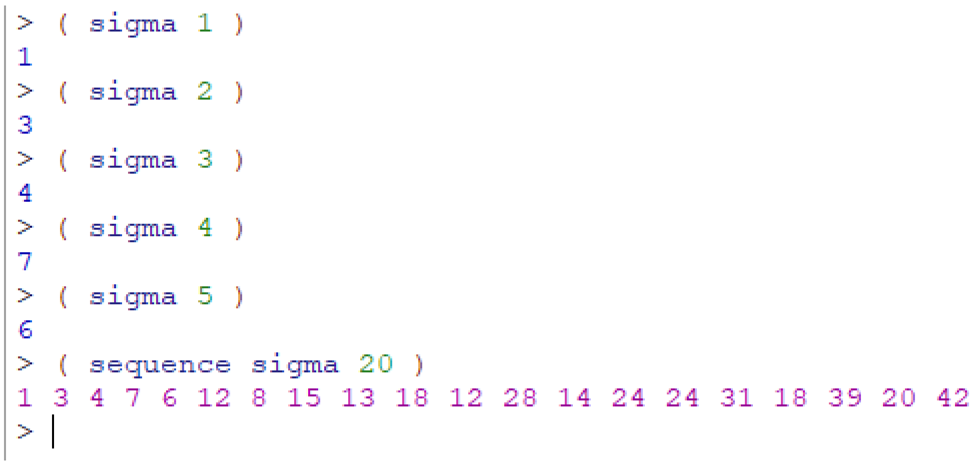
```
> ( sequence sigma 20 )
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42
> ( square 5 )
25
> ( square 10 )
100
> ( sequence square 15 )
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
> ( cube 2 )
8
> ( cube 3 )
27
> ( sequence cube 15 )
1 8 27 64 125 216 343 512 729 1000 1331 1728 2197 2744 3375
> 
```

Preliminary Demo



```
> ( triangular 1 )
1
> ( triangular 2 )
3
> ( triangular 3 )
6
> ( triangular 4 )
10
> ( triangular 5 )
15
> ( sequence triangular 20 )
1 3 6 10 15 21 28 36 45 55 66 78 91 105 120 136 153 171 190 210
>
```

Triangular Demo



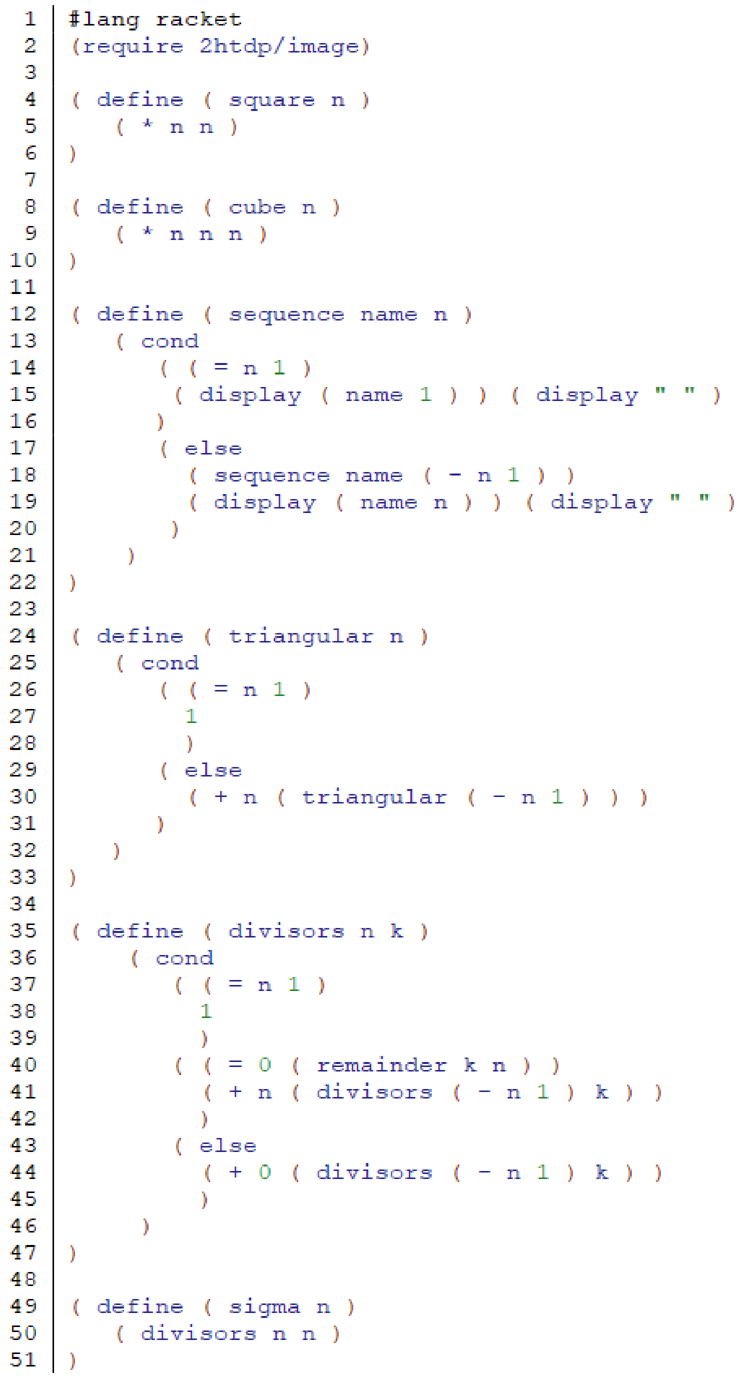
```
> ( sigma 1 )
1
> ( sigma 2 )
3
> ( sigma 3 )
4
> ( sigma 4 )
7
> ( sigma 5 )
6
> ( sequence sigma 20 )
1 3 4 7 6 12 8 15 13 18 12 28 14 24 24 31 18 39 20 42
> 
```

Sigma Demo

Task 3 asks that we create some simple mathematics functions to code popular patterns and sequences. These functions are trick because they are probably much simpler than someone would think at first for intricate patterns.

Number Sequence Code

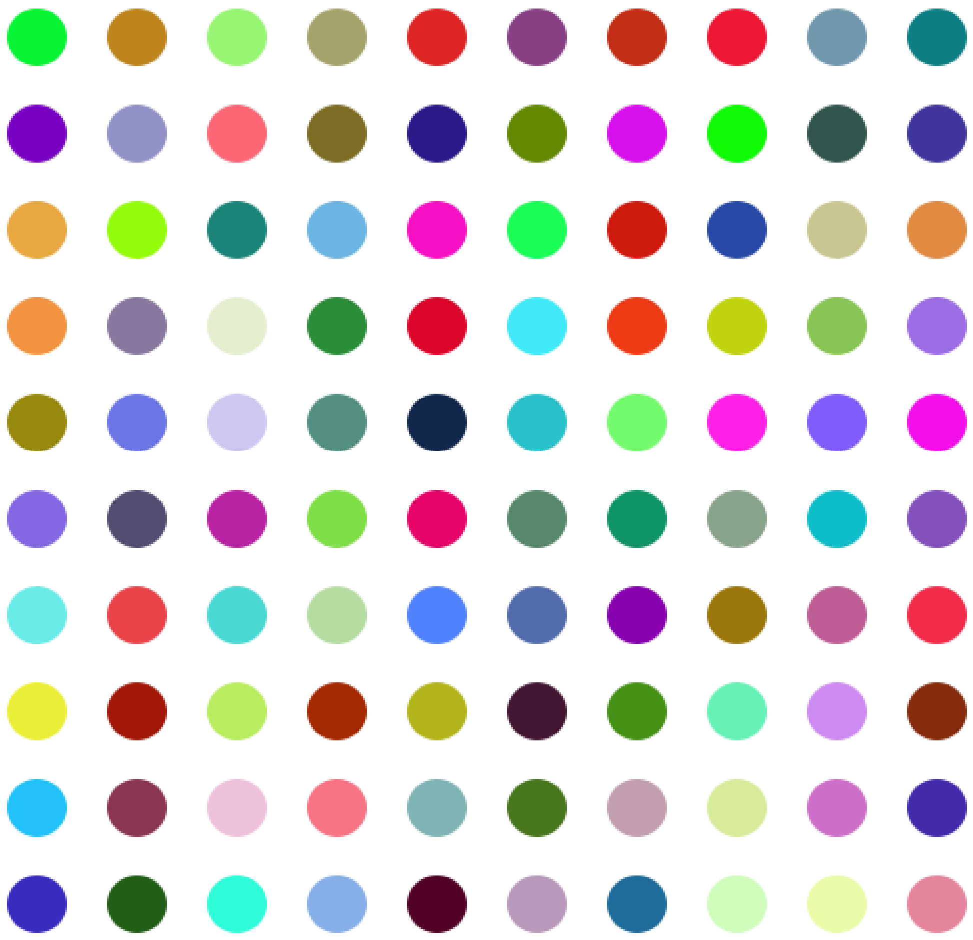
Like I stated above, the code is much simpler than some functional programming functions due to it’s proper use of recursion.



```
#lang racket
(require 2htdp/image)

( define ( square n )
   ( * n n )
)

( define ( cube n )
   ( * n n n )
)

( define ( sequence name n )
   ( cond
      ( ( = n 1 )
       ( display ( name 1 ) ) ( display " " )
      )
      ( else
        ( sequence name ( - n 1 ) )
        ( display ( name n ) ) ( display " " )
       )
    )
)

( define ( triangular n )
   ( cond
      ( ( = n 1 )
        1
        )
      ( else
        ( + n ( triangular ( - n 1 ) ) )
      )
   )
)

( define ( divisors n k )
    ( cond
       ( ( = n 1 )
         1
         )
       ( ( = 0 ( remainder k n ) )
         ( + n ( divisors ( - n 1 ) k ) )
         )
       ( else
         ( + 0 ( divisors ( - n 1 ) k ) )
         )
     )
)

( define ( sigma n )
   ( divisors n n )
)
```

Task 4 – Hirst Dots

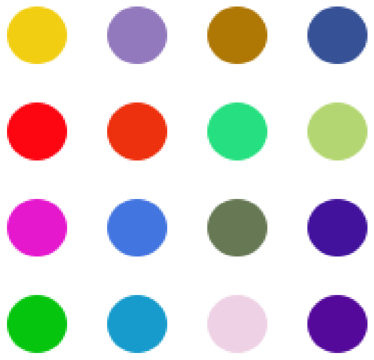
```
> ( hirst-dots 10 )
```



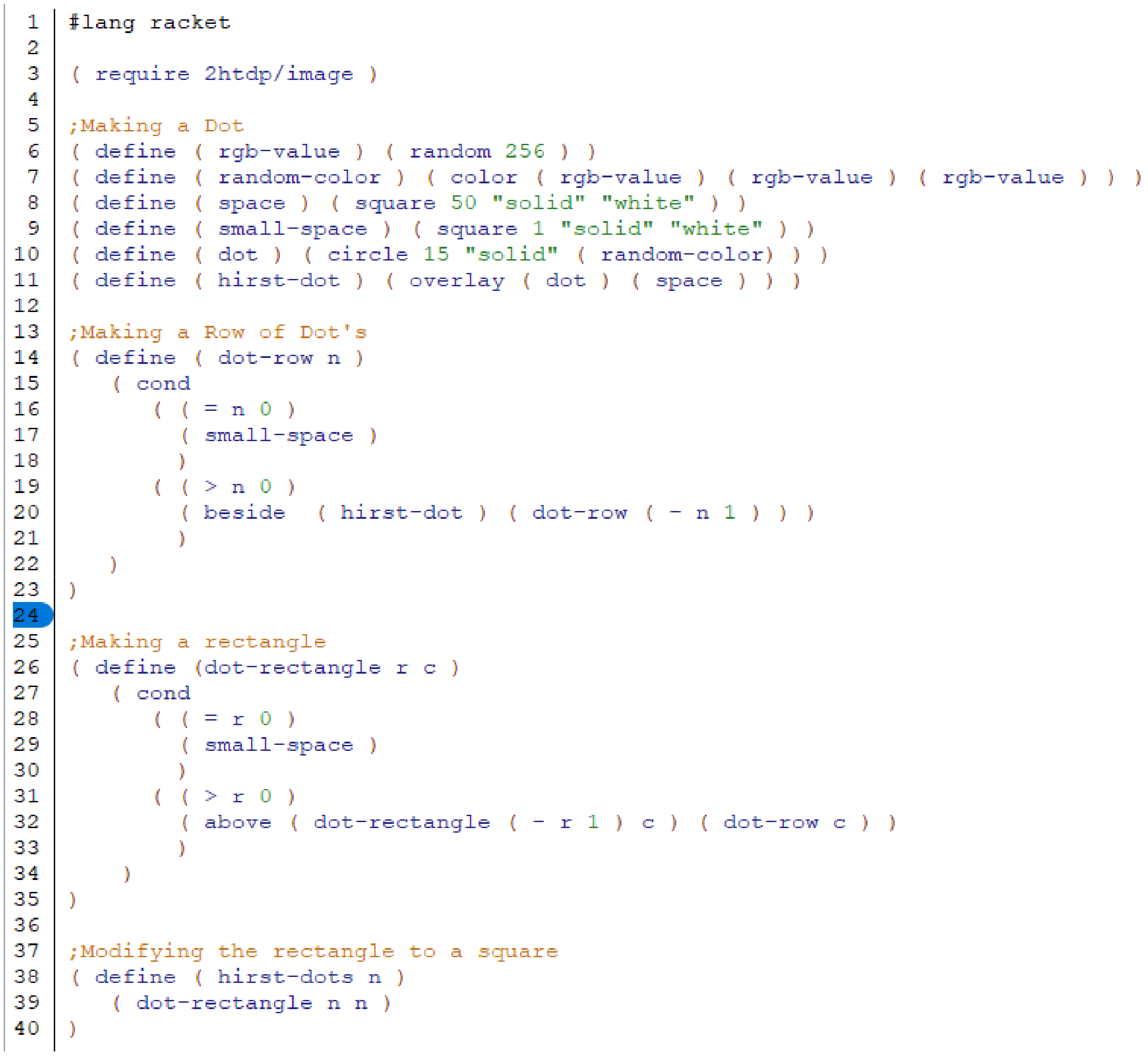
Hirst Dot Demo

This code emulates a twister board.

```
> ( hirst-dots 4 )
```



```
>
```



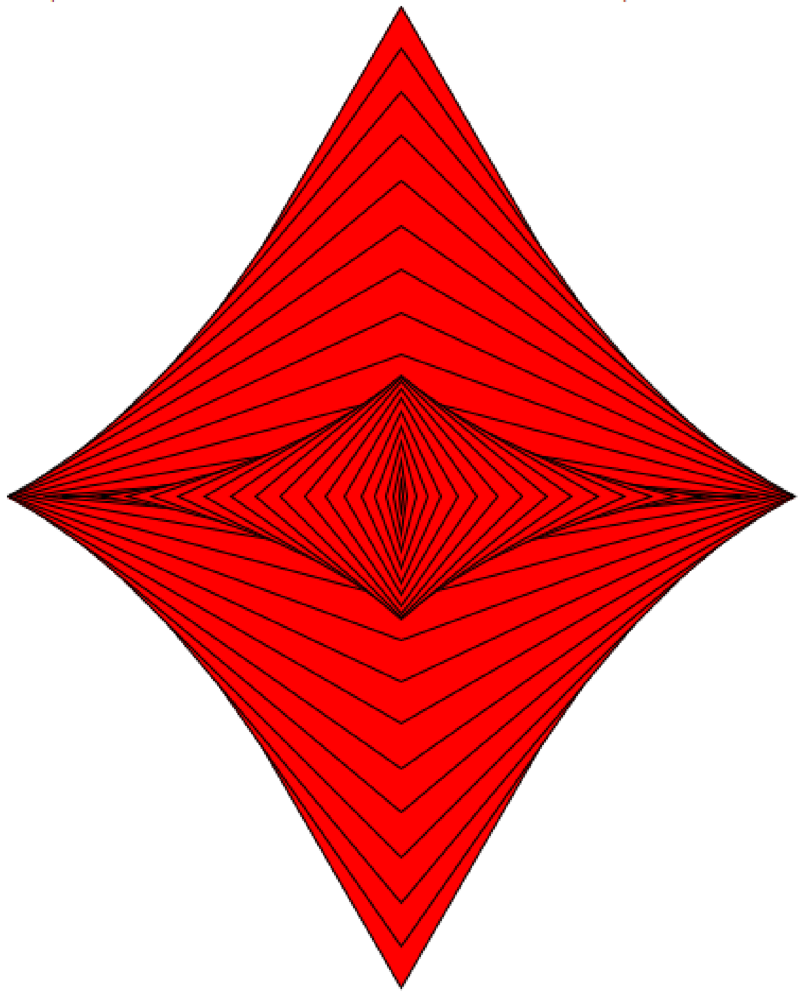
```
#lang racket

( require 2htdp/image )

;Making a Dot
( define ( rgb-value ) ( random 256 ) )
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )
( define ( space ) ( square 50 "solid" "white" ) )
( define ( small-space ) ( square 1 "solid" "white" ) )
( define ( dot ) ( circle 15 "solid" ( random-color) ) )
( define ( hirst-dot ) ( overlay ( dot ) ( space ) ) )

;Making a Row of Dot's
( define ( dot-row n )
   ( cond
      ( ( = n 0 )
        ( small-space )
        )
      ( ( > n 0 )
        ( beside  ( hirst-dot ) ( dot-row ( - n 1 ) ) )
        )
   )
)

;Making a rectangle
( define (dot-rectangle r c )
   ( cond
      ( ( = r 0 )
        ( small-space )
        )
      ( ( > r 0 )
        ( above ( dot-rectangle ( - r 1 ) c ) ( dot-row c ) )
        )
    )
)

;Modifying the rectangle to a square
( define ( hirst-dots n )
   ( dot-rectangle n n )
)
```

Hirst Dot Code

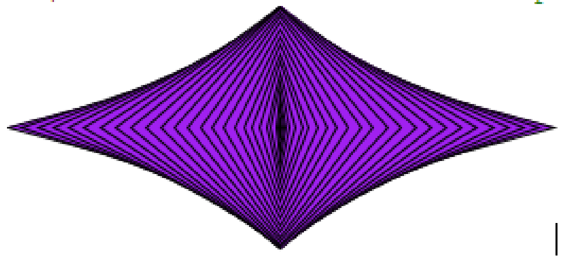
This code makes great use of recursion also. The only part of this code that irritates me is the fact that I felt like I had to create a function just to set the parameters for another function. It seems silly but I could not think of another solution.

Task 5 – Channeling Frank Stella

```
> ( nested-rhombus-one 300 30 "red" )
```



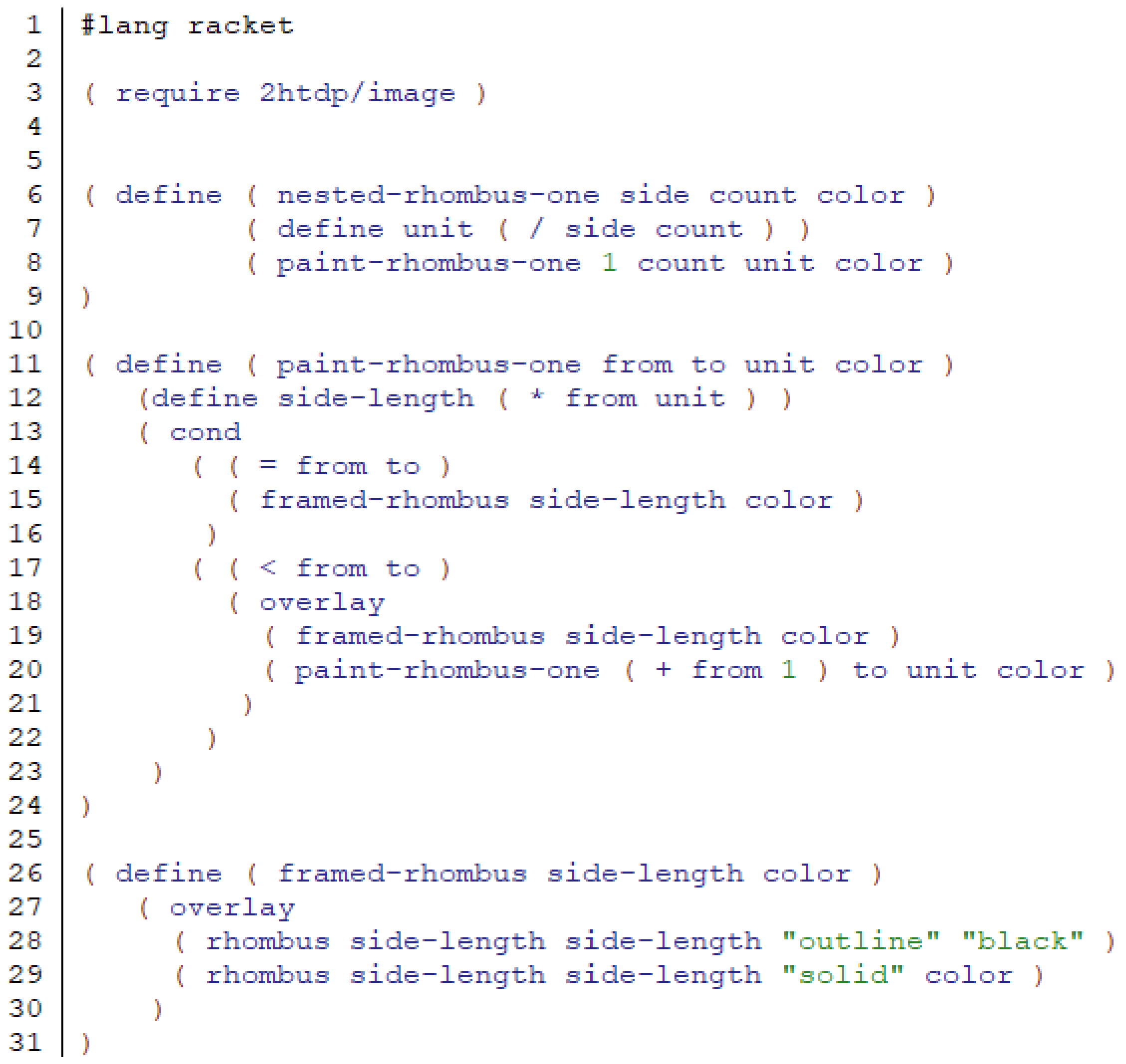
```
> ( nested-rhombus-one 150 25 "purple" )
```



```
>
```

Stella Image

I liked this task a lot. It’s result led me to think of it for my creation later on.



```
#lang racket

( require 2htdp/image )


( define ( nested-rhombus-one side count color )
          ( define unit ( / side count ) )
          ( paint-rhombus-one 1 count unit color )
)

( define ( paint-rhombus-one from to unit color )
   (define side-length ( * from unit ) )
   ( cond
      ( ( = from to )
        ( framed-rhombus side-length color )
       )
      ( ( < from to )
        ( overlay
          ( framed-rhombus side-length color )
          ( paint-rhombus-one ( + from 1 ) to unit color )
         )
       )
    )
)

( define ( framed-rhombus side-length color )
   ( overlay
     ( rhombus side-length side-length "outline" "black" )
     ( rhombus side-length side-length "solid" color )
    )
)
```

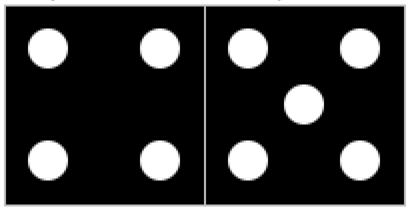
Stella Code

This code is a little confusing at first. After digesting it for a while it is interesting to see the simple effects a few lines of code can have on objects.

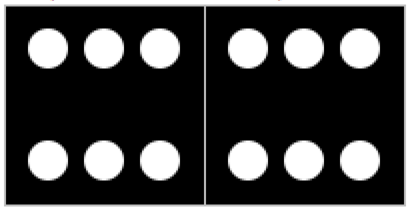
Task 6 – Dominos

Dominos Final Demo

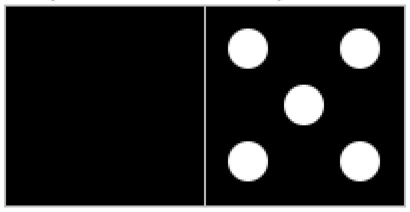
```
> ( domino 4 5 )
```



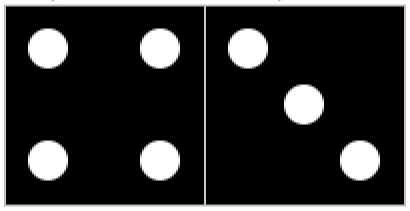
```
> ( domino 6 6 )
```



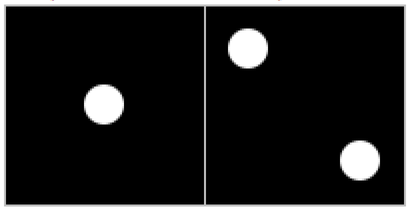
```
> ( domino 0 5 )
```



```
> ( domino 4 3 )
```

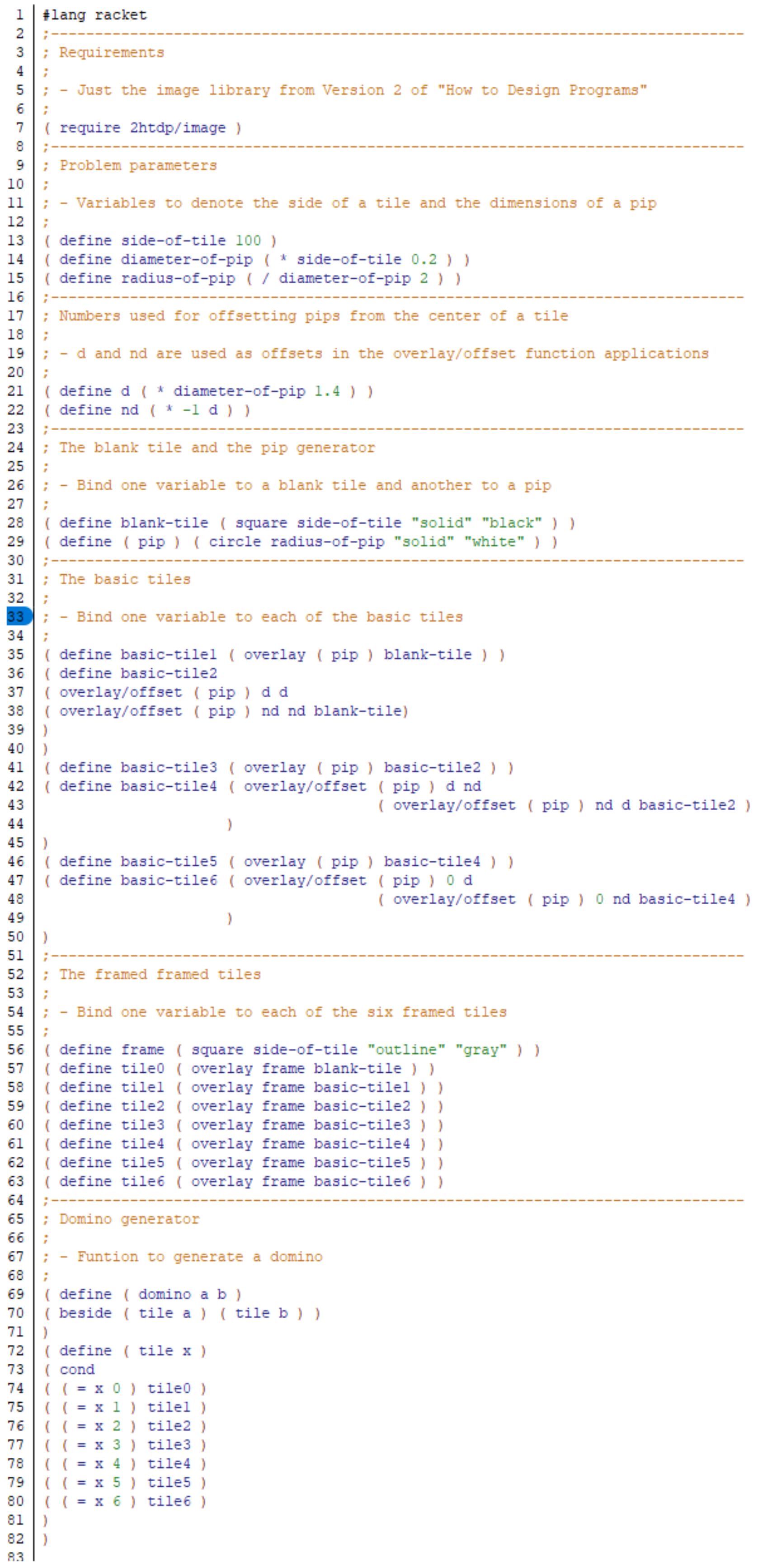


```
> ( domino 1 2 )
```



This task was fun to play around with some functions until I understood what they were doing better. After that it was a fairly simple task.

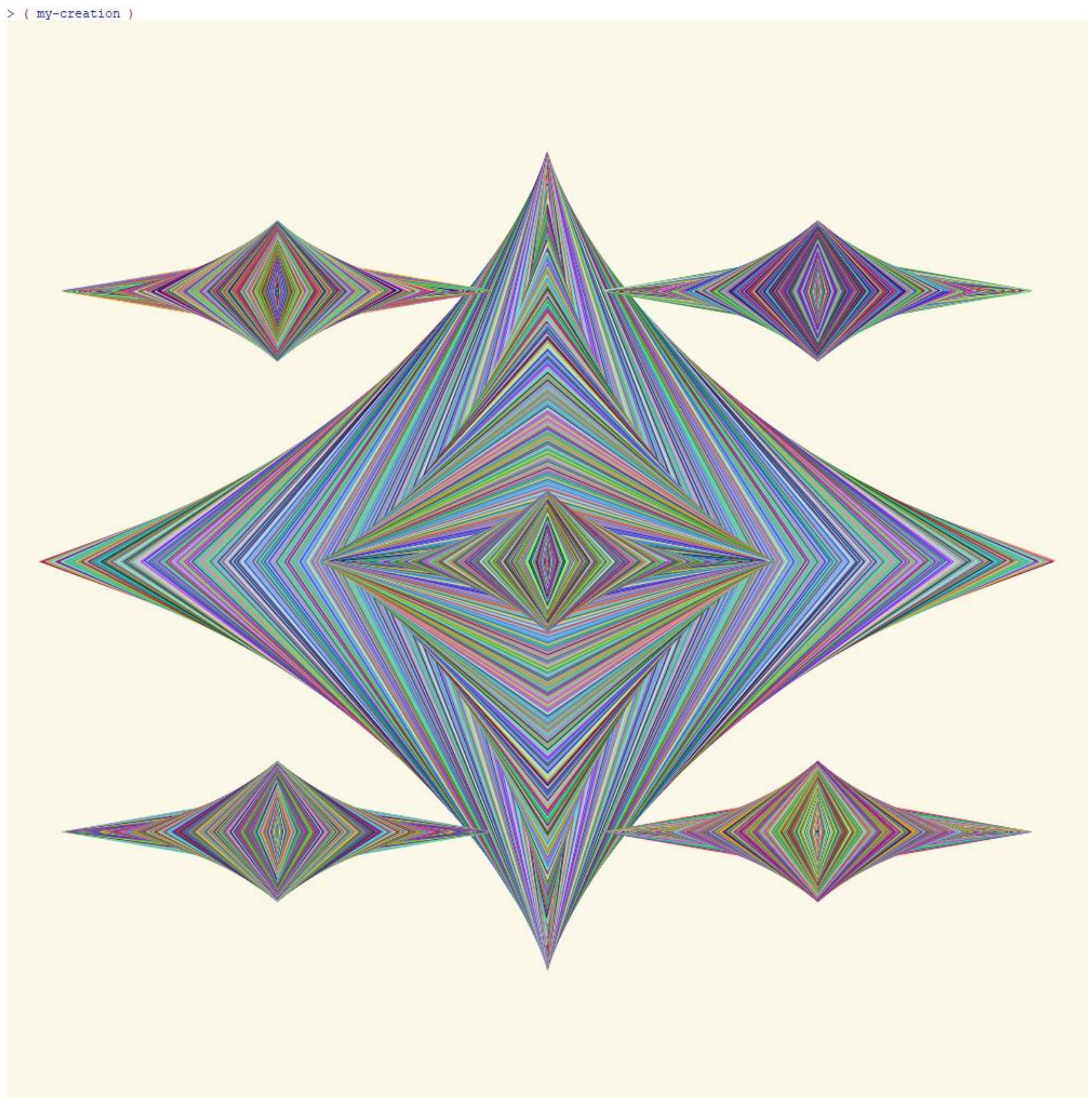
Dominos Code



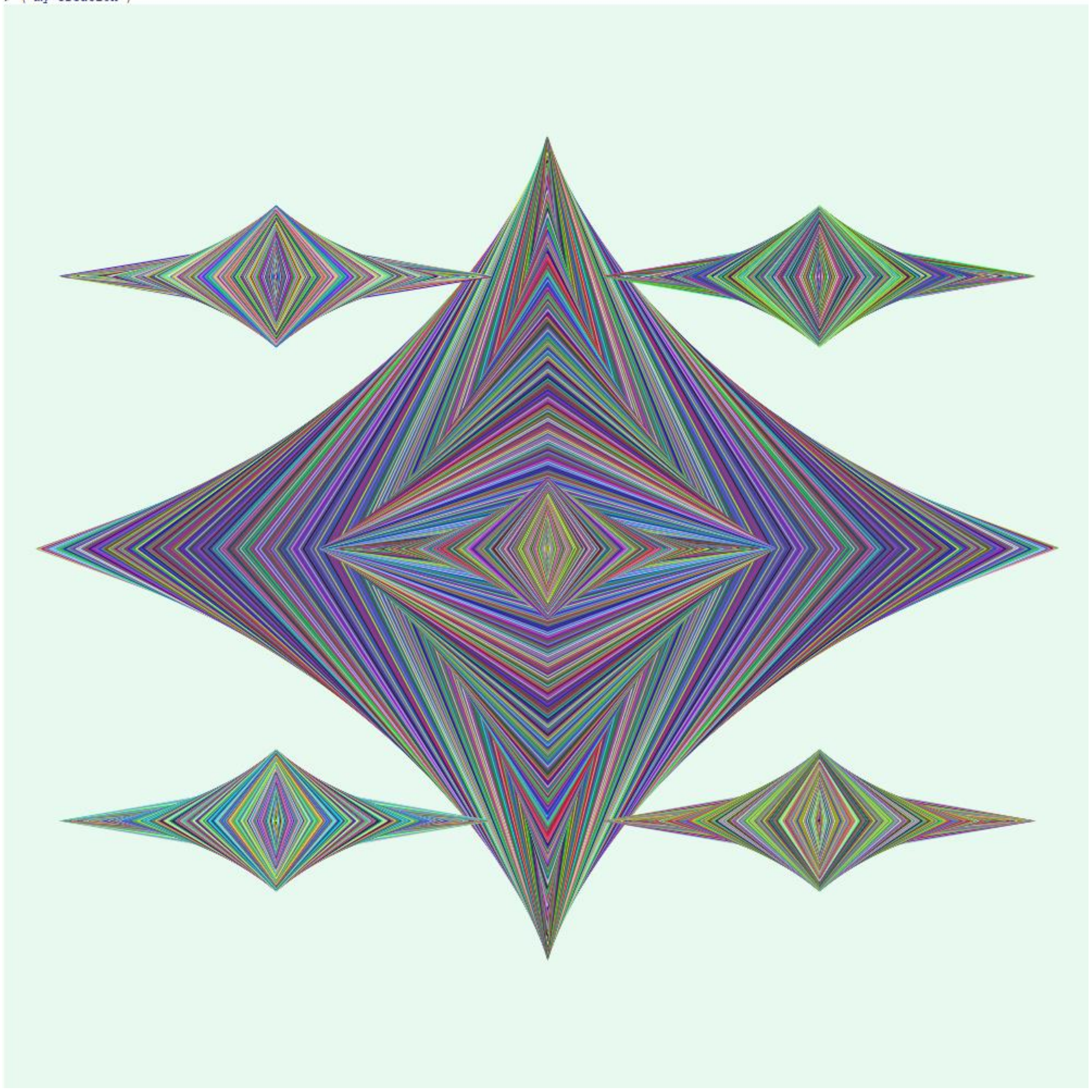
```
#lang racket
;--------------------------------------------------------------------------------
; Requirements
;
; - Just the image library from Version 2 of "How to Design Programs"
;
( require 2htdp/image )
;--------------------------------------------------------------------------------
; Problem parameters
;
; - Variables to denote the side of a tile and the dimensions of a pip
;
( define side-of-tile 100 )
( define diameter-of-pip ( * side-of-tile 0.2 ) )
( define radius-of-pip ( / diameter-of-pip 2 ) )
;--------------------------------------------------------------------------------
; Numbers used for offsetting pips from the center of a tile
;
; - d and nd are used as offsets in the overlay/offset function applications
;
( define d ( * diameter-of-pip 1.4 ) )
( define nd ( * -1 d ) )
;--------------------------------------------------------------------------------
; The blank tile and the pip generator
;
; - Bind one variable to a blank tile and another to a pip
;
( define blank-tile ( square side-of-tile "solid" "black" ) )
( define ( pip ) ( circle radius-of-pip "solid" "white" ) )
;--------------------------------------------------------------------------------
; The basic tiles
;
; - Bind one variable to each of the basic tiles
;
( define basic-tile1 ( overlay ( pip ) blank-tile ) )
( define basic-tile2
( overlay/offset ( pip ) d d
( overlay/offset ( pip ) nd nd blank-tile)
)
)
( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )
( define basic-tile4 ( overlay/offset ( pip ) d nd
                                      ( overlay/offset ( pip ) nd d basic-tile2 )
                     )
)
( define basic-tile5 ( overlay ( pip ) basic-tile4 ) )
( define basic-tile6 ( overlay/offset ( pip ) 0 d
                                      ( overlay/offset ( pip ) 0 nd basic-tile4 )
                     )
)
;--------------------------------------------------------------------------------
; The framed framed tiles
;
; - Bind one variable to each of the six framed tiles
;
( define frame ( square side-of-tile "outline" "gray" ) )
( define tile0 ( overlay frame blank-tile ) )
( define tile1 ( overlay frame basic-tile1 ) )
( define tile2 ( overlay frame basic-tile2 ) )
( define tile3 ( overlay frame basic-tile3 ) )
( define tile4 ( overlay frame basic-tile4 ) )
( define tile5 ( overlay frame basic-tile5 ) )
( define tile6 ( overlay frame basic-tile6 ) )
;--------------------------------------------------------------------------------
; Domino generator
;
; - Funtion to generate a domino
;
( define ( domino a b )
( beside ( tile a ) ( tile b ) )
)
( define ( tile x )
( cond
( ( = x 0 ) tile0 )
( ( = x 1 ) tile1 )
( ( = x 2 ) tile2 )
( ( = x 3 ) tile3 )
( ( = x 4 ) tile4 )
( ( = x 5 ) tile5 )
( ( = x 6 ) tile6 )
)
)

```

Task 7 – Creation

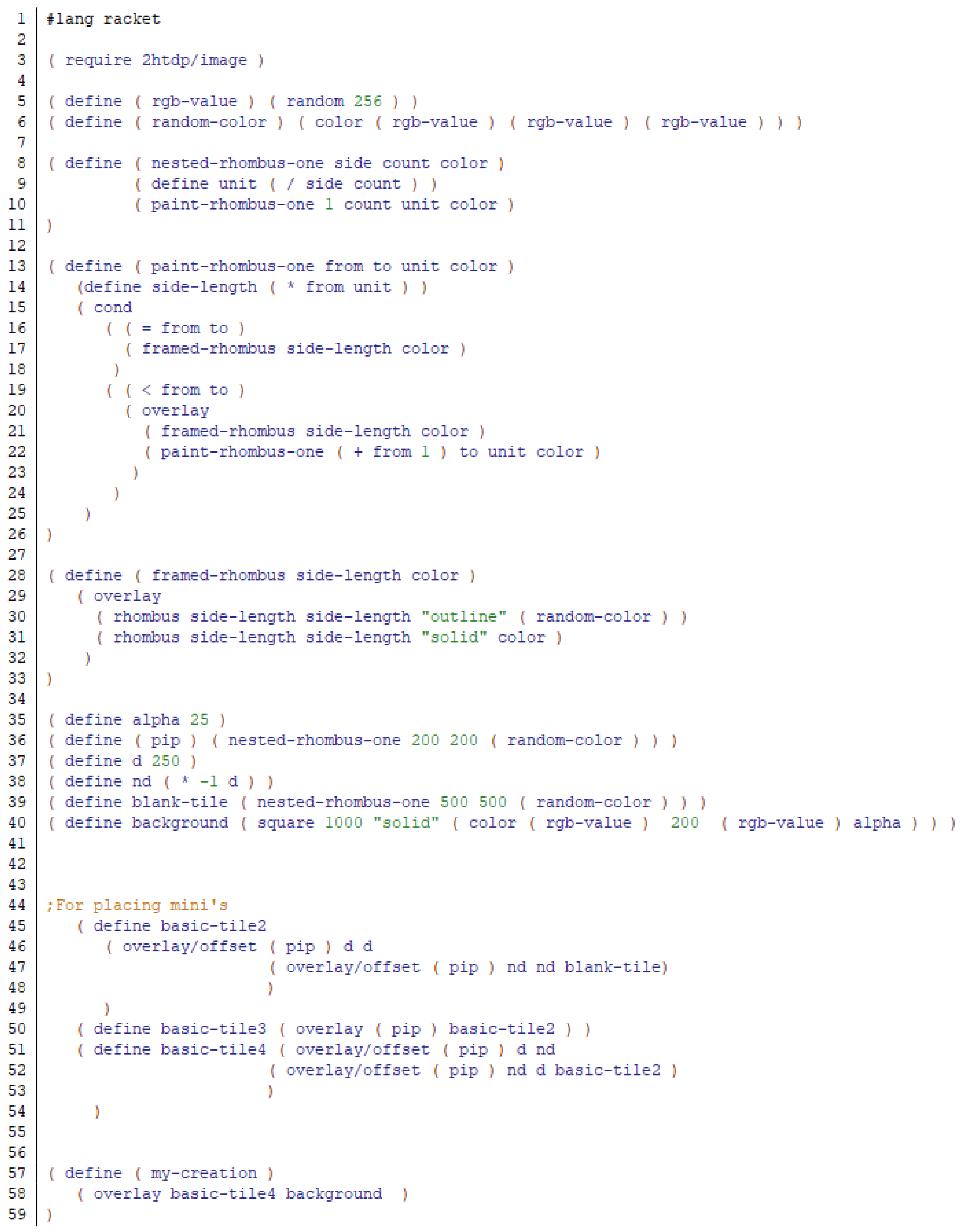
My creation Image



```
> ( my-creation )
> ( my-creation )
```



This image was inspired from the Stella task from above. I liked how the Rhombus seemed to become recursive and change it's direction after a certain size. If I spent more time I would try have the function display the image multiple times to create a color changing effect and/or I would change the size's and positions of the miniature rhombus'.



```
#lang racket

( require 2htdp/image )

( define ( rgb-value ) ( random 256 ) )
( define ( random-color ) ( color ( rgb-value ) ( rgb-value ) ( rgb-value ) ) )

( define ( nested-rhombus-one side count color )
         ( define unit ( / side count ) )
         ( paint-rhombus-one 1 count unit color )
)

( define ( paint-rhombus-one from to unit color )
   (define side-length ( * from unit ) )
   ( cond
      ( ( = from to )
        ( framed-rhombus side-length color )
       )
      ( ( < from to )
        ( overlay
          ( framed-rhombus side-length color )
          ( paint-rhombus-one ( + from 1 ) to unit color )
         )
       )
    )
)

( define ( framed-rhombus side-length color )
   ( overlay
     ( rhombus side-length side-length "outline" ( random-color ) )
     ( rhombus side-length side-length "solid" color )
    )
)

( define alpha 25 )
( define ( pip ) ( nested-rhombus-one 200 200 ( random-color ) ) )
( define d 250 )
( define nd ( * -1 d ) )
( define blank-tile ( nested-rhombus-one 500 500 ( random-color ) ) )
( define background ( square 1000 "solid" ( color ( rgb-value )  200  ( rgb-value ) alpha ) ) )



;For placing mini's
   ( define basic-tile2
      ( overlay/offset ( pip ) d d
                       ( overlay/offset ( pip ) nd nd blank-tile)
                       )
      )
   ( define basic-tile3 ( overlay ( pip ) basic-tile2 ) )
   ( define basic-tile4 ( overlay/offset ( pip ) d nd
                       ( overlay/offset ( pip ) nd d basic-tile2 )
                       )
     )


( define ( my-creation )
   ( overlay basic-tile4 background  )
)
```

My creation Code

The thing I liked about this code the most, honestly, I that it is only a few alterations to the original code we had played with already. By tweaking some parameters and using functions from several tasks I created my creation. And while it is similar code to what we used, it looks drastically different.